Wykonał: Jakub Nowakowski

1. **Cele Laboratorium**
   Laboratorium ma na celu przedstawienie algorytmu zamiatania sprawdzającego przecięcia zadanego zbioru odcinków oraz pokazać jak istotne przy wydajnej implementacji algorytmu są wykorzystane struktury danych.

2. **Konfiguracja stanowiska**
   Procesor: intel Corei7-8750H
   Wersja Python'a: Python 3.6

3. **Wstęp teoretyczny**
   Algorytm zamiatania wykorzystany do sprawdzania przecięć pomiędzy odcinkami wykorzystuje dwie struktury do poprawnego działania: strukturę stanu i strukturę zdarzeń. Struktura stanu przechowuje informacje o tym które odcinki są aktualnie analizowane, a struktura zdarzeń przechowuje punkty, na których będzie zatrzymywać się „miotła". Algorytm swoją wydajność zawdzięcza temu, że nie sprawdza wszystkich par odcinków, a jedynie nowo pojawiających się sąsiadów w strukturze stanu. Złożoność tego algorytmu zamiatania to: O((P+n)*logn) składa się na nią:
   - inicjalizacja struktury zadaszeń O(nlogn) (posortowanie końców i początków odcinków względem współrzędnej x-owej)
   - aktualizacja struktury stanów odbywa się P+2n razy przy optymalnej strukturze jednorazowo O(logn) czyli całkowita złożoność aktualizacji struktury stanów to O((P+n)*logn)
   - aktualizacja struktury zdarzeń odbywa się dla każdego punktu przecięcia, a przy optymalnej strukturze każda aktualizacja odbywa się w O(logn) czyli razem O(P*logn)

   Dla porównania algorytm brutalny sprawdzający każdy odcinek z każdym ma złożoność O(n^2)

4. **Opis działania programu**

   Program działa w następujący sposób:

   - Najpierw następuje inicjalizacja struktury zdarzeń punktami końców i początków odcinków posortowanymi względem współrzędnej x-owej
   - Następnie miotła przemieszcza się po strukturze zdarzeń, gdy napotka wierzchołek początkowy dodaje odpowiadający mu odcinek, dla wierzchołków kończących usuwa odpowiadający mu odcinek ze struktury stanów, dla wierzchołków odpowiadających przecięciu odcinków, odcinki zamieniane są miejscami w strukturze stanów.

- Gdy w strukturze stanów pojawia się nowa para sąsiadów, sprawdzane jest czy sąsiedzi się nie przecinają, a jeśli tak i nie było to uwzględnione wcześniej dodawany jest punkt ich przecięcia do struktury zdarzań.

Dla programu sprawdzającego czy jakiekolwiek dwa odcinki się przecinają, zasada działania jest analogiczna, lecz przy znalezieniu pierwszego punktu przecięcia zwraca on informacje o jego wystąpieniu, dlatego tez nie ma konieczności modyfikowania struktury zdarzeń.

5. Wykorzystane struktury danych

- Reprezentacja punktów
  Punkty reprezentowane są za pomocą klasy Vector2d która przechowuje informacje o typie wierzchołka (początek odcinka, koniec odcinka, przecięcie odcinków), przechowuje index linii, do której/których (dla wierzchołka reprezentującego przecięcie) należy. Struktura punktu umożliwia jego haszowane, wykorzystując do tego nie współrzędna, ale indeksy linii, które w danym punkcie się przecinają, dzięki temu mamy gwarancje, że jedno przecięcie nie zostanie zliczone dwa razy, bo dwa odcinki nie współliniowe, mogą przecinać się tylko w jednym punkcie.
- Reprezentacja odcinka
  Odcinek reprezentowany jest za pomocą klasy Line która przechowuje informacje o początku odcinka, jego końcu, oraz równanie prostej na której leży dany odcinek. Ponadto klasa Line przechowuje informacje o aktualnej współrzędnej x-owej, która jest wykorzystywana w komparatorze porównującym odcinki na podstawie współrzędnej y-kowej przy wstawianiu do struktury stanów.
- Struktury stanu i zdarzeń
  Struktury stanu zarówno w programie sprawdzającym czy istnieją jakiekolwiek przecięcia i znajdującym wszystkie przecięcia zostały zaimplementowane w ten sam sposób. W obu przypadkach są reprezentowane przez obiekt klasy sorterdset z modułu blist. Powody wyboru tej struktury danych:
  - Prostota użytku. Struktura ta nie wymaga implementacji haszowania obiektu reprezentującego odcinek, a jedynie komparatora porównującego wstawiane elementy.
  - Gwarancja posortowania elementów wewnątrz zbioru
  - Eliminacja powtórzeń w zbioru
  - Szybkie wstawianie elementów bazujące na wyszukiwaniu binarnym mające złożoność O(logn)

- Szybkie usuwanie konkretnego elementu O(logn)
-

W przypadku programu znajdującego wszystkie przecięcia struktura zdarzeń jest zaimplementowana również jako sortedset, ponieważ ta struktura zapewnia nam:

- Utrzymanie posortowania w strukturze
- Szybkie dodawanie nowego elementu O(logn)
- Szybkie usuwanie elementów O(logn)

W programie określającym tylko czy przecięcie odcinków wystąpiło wystarczającą strukturą do przechowywania zdarzeń jest zwykła lista, ponieważ nie będziemy potrzebować dodawania punktów do już posortowanej struktury.

- Struktura przechowująca punkty przecięcia w programie szukającym wszystkich takich punktów.
  Do wybory tej struktury został wybrany set z biblioteki standardowej, przy wykorzystaniu haszowania punktu opisanego wyżej, umożliwia nam:
  - Dodawanie punktów do wyniku w czasie jednostkowym
  - Sprawdzenie wystąpienia punktu w wyniku w czasie jednostkowym
  - Eliminacje ewentualnych powtórzeń
- Struktura przechowująca zbiór odcinków
  Odcinki przekazywane są do programu w postaci listy, do której referencje przechowywane są później wewnątrz poszczególnych zdarzeń, oraz w samej strukturze stanów. Dzięki przechowywaniu w strukturach referencji do oryginalnych odcinków minimalizujemy ryzyko błędów wynikających z precyzji obliczeń oraz zyskujemy dostęp do konkretnych odcinków w czasie jednostkowym.

6. Obsługa zdarzeń w programie

Zdarzenia w programie reprezentowane są wewnątrz klasy opisującej punkt.

Obsługa zdarzeń dla programu szukającego wszystkich przecięć i programu szukającego jakichkolwiek przecięć jest taka sama dla zdarzenia początku odcinka i końca odcinka. Różnica między programami pojawia się w monecie wykrycia przecięcia odcinków, program szukający dowolnego przecięcia kończy wtedy działanie, drugi program musi dodać do struktury zdarzeń punkt odpowiadający przecięciu, a następnie punkt ten dodatkowo rozpatrzyć.

- Zdarzenie startu odcinka
  - Ustawiamy x dla komparatora jako współrzędną x-ową aktualnego punktu startu odcinka
  - Dodajemy zaczynający się odcinek do struktury stanów
  - Sprawdzamy czy nowo dodany odcinek ma punkty przecięcia z swoimi sąsiadami w strukturze stanów
- Zdarzenie końca odcinka
  - Odnajdujemy kończący się odcinek w strukturze stanu
  - Po odnalezieniu odcinka w strukturze stanu odnajdujemy sąsiadów kończącego się odcinka (o ile istnieją)
  - Jeżeli sąsiedzi istnieją sprawdzamy czy nie przecinają się
  - Usuwamy kończący się odcinek ze struktury stanów

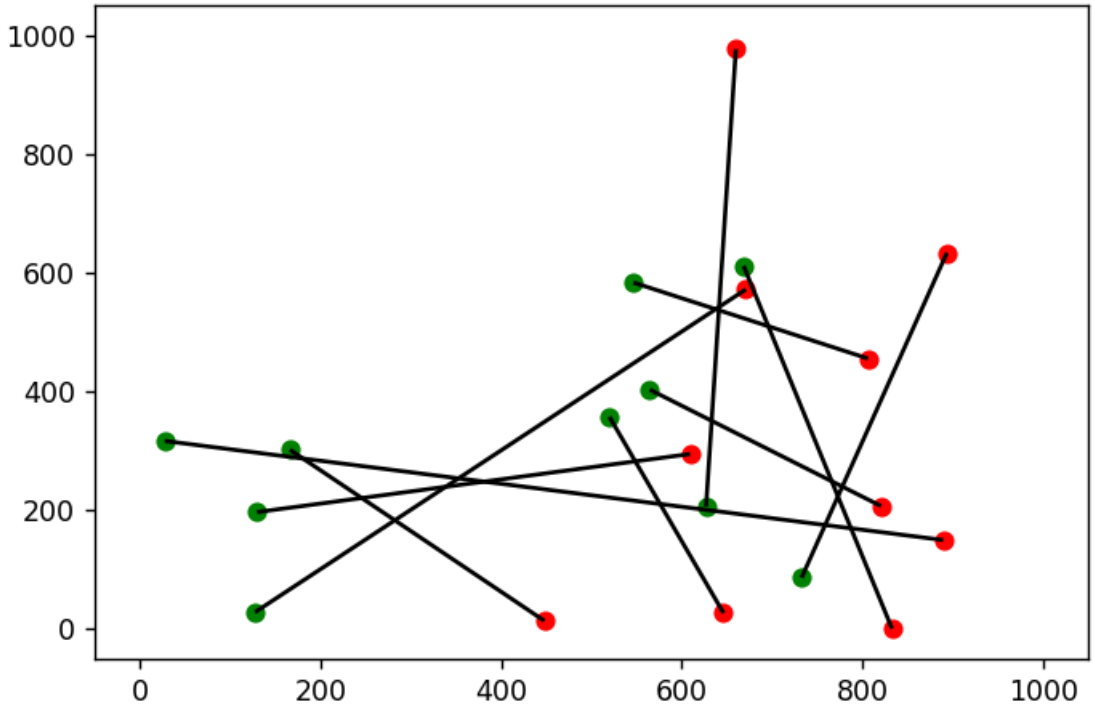
- Zdarzenie przecięcia odcinków
  - Pobieramy ze zdarzenia referencje do przecinających się odcinków
  - Usuwamy odcinki ze struktury stanów
  - Zmieniamy aktualne x w komparatorze odcinków o minimalne delta x w prawo (dzięki temu współrzędne y obu odcinków nie będą już takie same)
  - Dodajemy na nowo odcinki do struktury stanów (te dwa kroki zamieniają odcinki miejscami w strukturze stanów)
  - Pobieramy nowe id obu odcinków w strukturze stanów, a następnie sprawdzamy istnienie ich sąsiadów zewnętrznych i sprawdzamy, czy odcinki przecinają się z nimi

7. Generowanie zestawów testowych

Generator losowych odcinków korzysta z funkcji uniform w celu wygenerowania losowych punktów z podanego zakresu. Zakres punktów z którego mają być generowane punkty początkowe i końcowe prostych jest opisany przez lewy dolny i prawy górny wierzchołek prostokąta, w którym mają znajdować się odcinki. Generator generuje zadaną liczbę współrzędnych eliminując powtarzające się X owe umieszczając je wcześniej w secie, dzięki temu nie pojawiają się odcinki pionowe ani o takich samych końcach.

Przykładowy wynik generowania 10 losowych odcinków:

	ID	współczynnik kierunkowy	b	punkt startu	punkt końca
0	0	-3.714299	3094.465779	(668.516428268956, 611.3959588875538)	(832.8748541271764, 0.9196420198652744)
1	1	0.203950	170.598482	(129.89402965778396, 197.09039742560662)	(609.2489426726154, 294.85493535019856)
2	2	-0.492280	852.395330	(546.5821561522538, 583.3240637898432)	(806.2739268933173, 455.4830927123641)
3	3	-2.612148	1715.425230	(519.9401943029732, 357.2647005444908)	(646.332370236747, 27.10968174796802)
4	4	-1.021908	471.796059	(166.18646058828378, 301.9687786863782)	(447.8526865368044, 14.131797262391444)
5	5	23.325476	-14417.568332	(626.9576977297849, 206.5185162188412)	(659.9504515646905, 976.0902101371355)
6	6	-0.765070	835.091271	(564.7046344210038, 403.0525475984873)	(820.6697617297717, 207.221240034119)
7	7	-0.193439	322.128449	(27.346939988941333, 316.83848930856306)	(889.5310946474325, 150.05859619055138)
8	8	3.397065	-2402.328238	(732.7202099491228, 86.77016649396107)	(893.6155550050303, 633.3421612303016)
9	9	0.999545	-98.840828	(127.5453985020325, 28.64652906610199)	(670.9488059150467, 571.8026517303656)

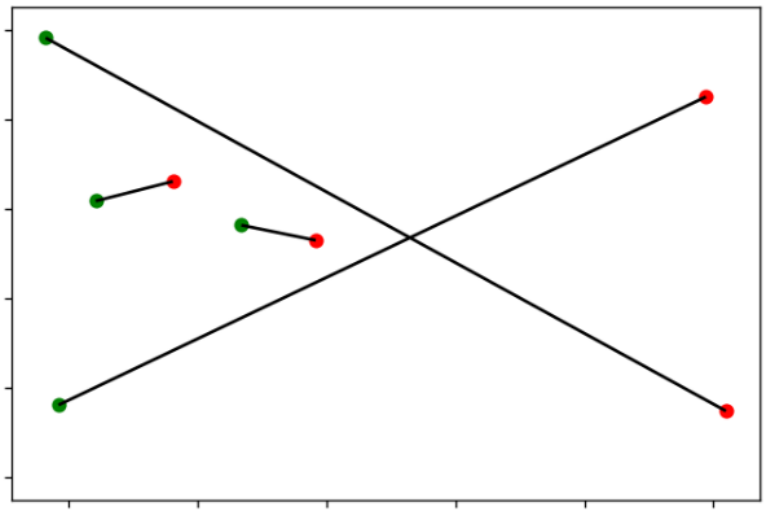


8. Dodawanie odcinków za pomocą myszki przez użytkownika

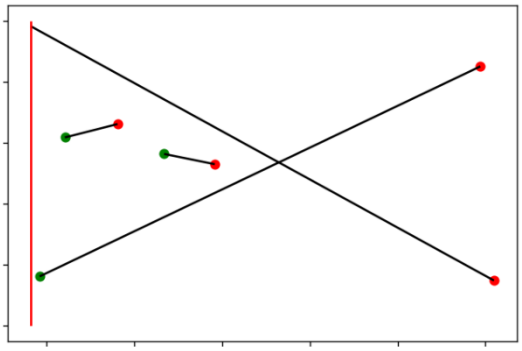
Program umożliwia również dodawanie docinków ręcznie przez użytkownika. Aby to zrobić należy kliknąć przycisk dodaj linie i wprowadzać odcinki zaznaczając ich końce i początki. Następnie program pobiera te odcinki i konwertuje je na obiekty klasy linę za pomocą zaimplementowanej do tego funkcji getLines().

9. Wizualizacja działania algorytmu i testy

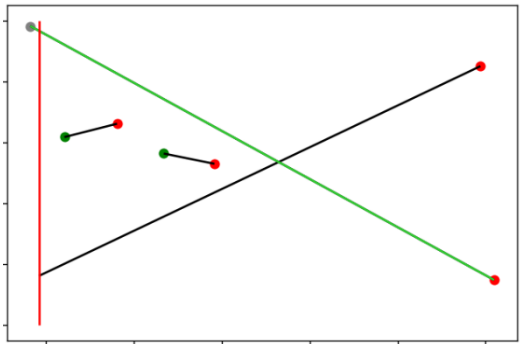
Wizualizacja działania algorytmu na przykładowym zbiorze, który pokazuje sytuację, w której jeden punkt przecięcia mógłby być zliczony klika razy, ponieważ dwa odcinki stają się swoimi sąsiadami więcej niż raz, jednak opisane wyżej haszowane punktów, wykorzystujące informacje które odcinki przecinają się w danym punkcie eliminuje taki błąd.



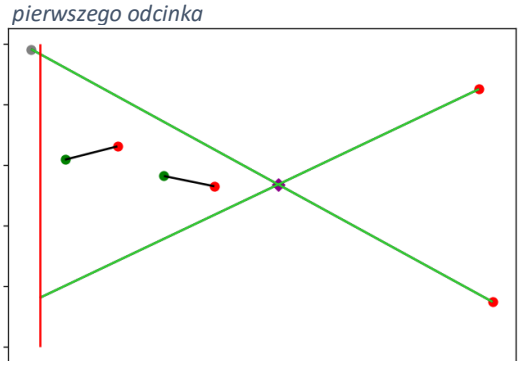
Rysunek 1 Zestaw testowy



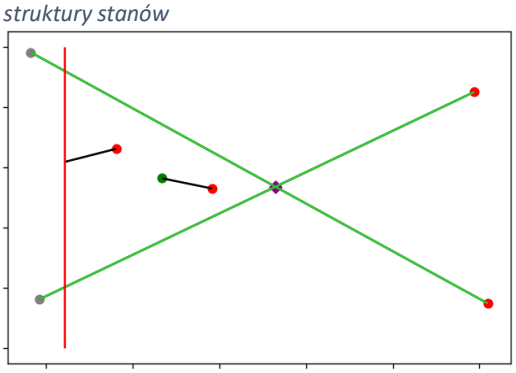
Rysunek 2 Pierwsza pozycja miotły, aktywacja pierwszego odcinka



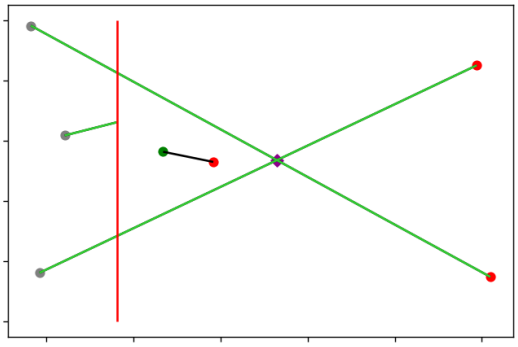
Rysunek 3 Początek drugiego odcinka, dodanie go do struktury stanów



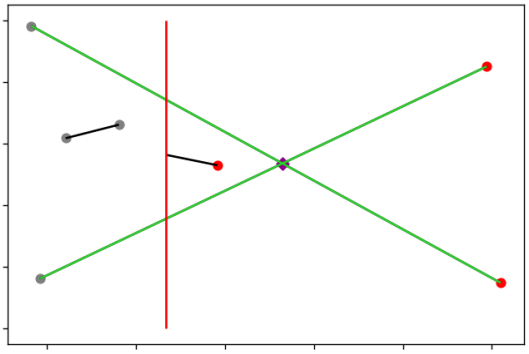
Rysunek 4 Zostaje wykryte przecięcie (romb na środku obrazka)



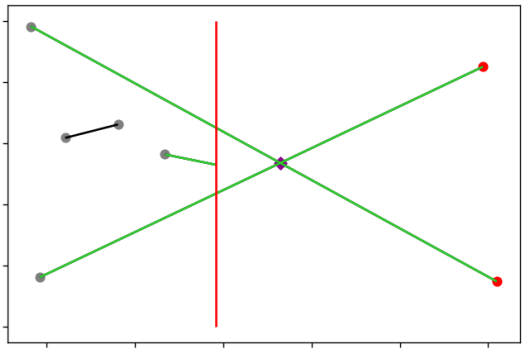
Rysunek 5 Dodanie kolejnego odcinka



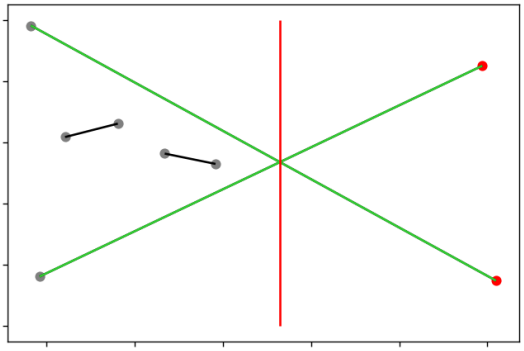
Rysunek 6 Dezaktywacja odcinka



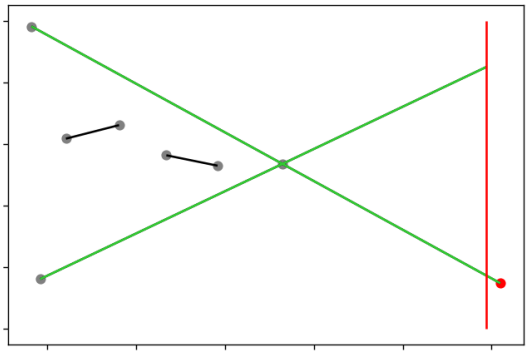
Rysunek 7 Dodanie odcinka



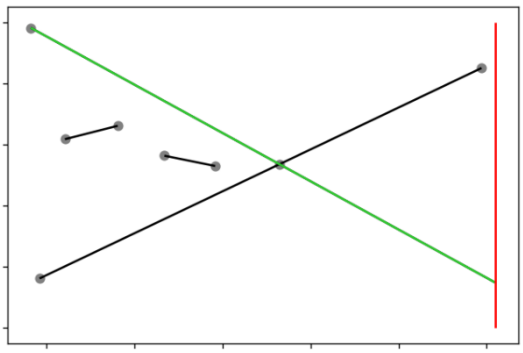
Rysunek 8 Dezaktywacja odcinka



Rysunek 9 Rozpatrzenie punktu przecięcia, zamiana miejsc w strukturze stanów



Rysunek 10 Dezaktywacja odcinka



Rysunek 11 Dezaktywacja ostatniego odcinka

	ID	współczynnik kierunkowy	b	punkt startu	punkt końca
0	0	0.684327	173.261756	(-15.99466877598914, 162.31617647058823)	(988.6424279982044, 849.8161764705883)
1	1	-0.788536	952.898704	(-35.95434619534399, 981.2500000000002)	(1019.6908150949786, 148.83578431372547)
2	2	0.365832	602.036448	(41.66662154659147, 617.2794117647061)	(161.4246860627205, 661.0906862745098)
3	3	-0.292232	641.639944	(267.8762989659464, 563.357843137255)	(383.1988796111076, 529.6568627450981)

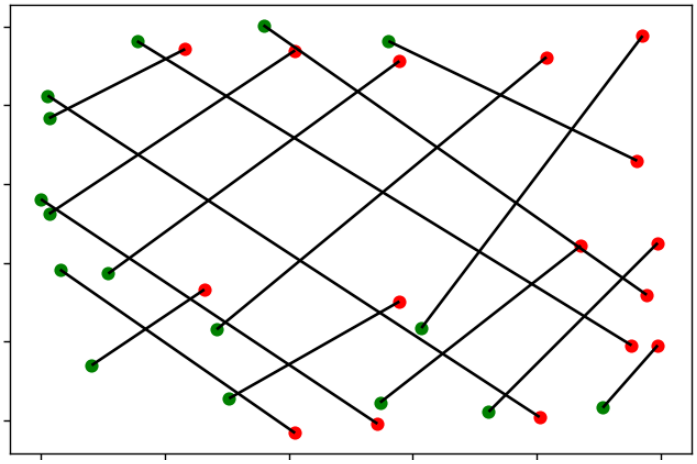
Rysunek 12 Proste na rysunku

	X	Y	ID Pierwszego odcinka	ID drugiego odcinka
0	529.334432	535.499446	0	1

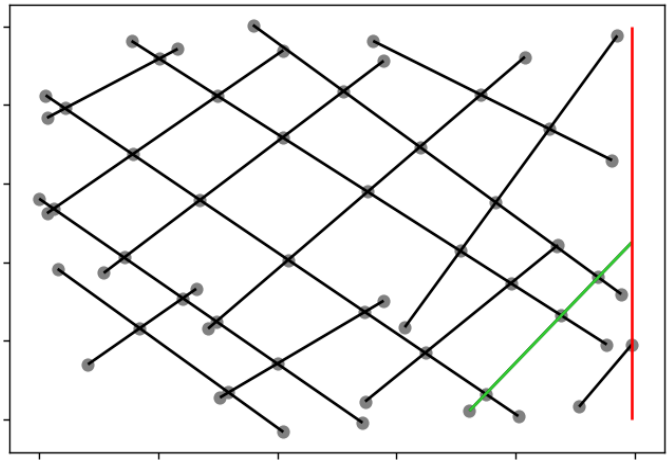
Rysunek 13 Znalezione punkty przecięcia

Inne zestawy testowe:

Zestaw sprawdzający przecięcia, gdy na jednym odcinku występuje ich kilka. Jak widać program znalazł je wszystkie.

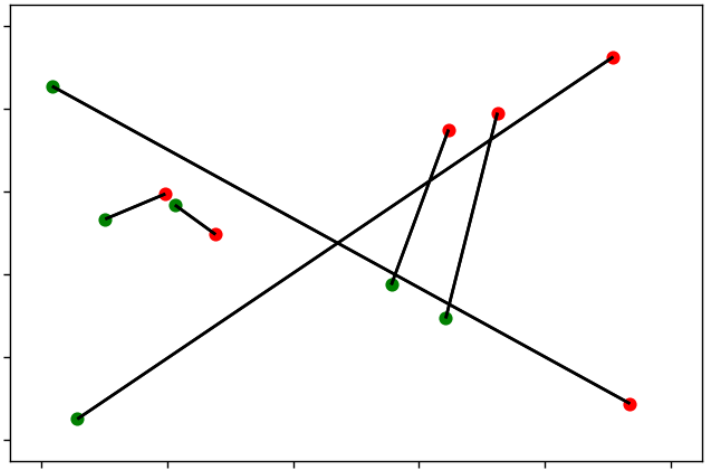


Rysunek 2 Zestaw testowy 2

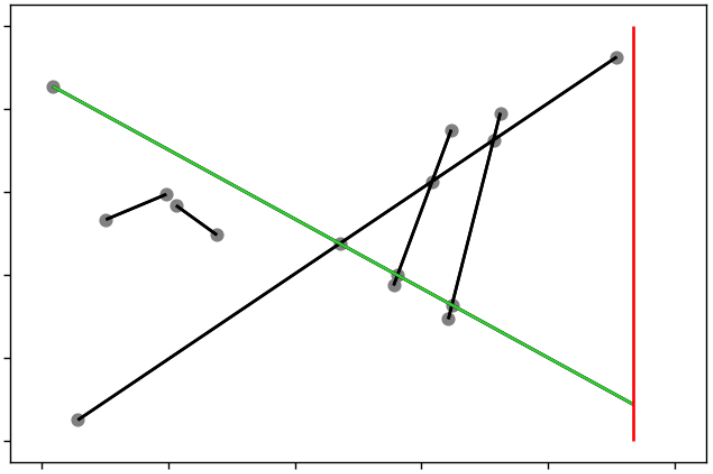


Rysunek 3 Wynik działania programu

Zestaw sprawdzający przypadek 1 i 2 jednocześnie, czy punkt jest liczony wielokrotnie, oraz czy wykrywane jest kilka przecięć na jednym odcinku.



Rysunek 5 Zestaw testowy 3



Rysunek 4 Wynik działania programu

	ID	współczynnik kierunkowy	b	punkt startu	punkt końca
0	0	-0.833215	867.857937	(17.27146025626891, 853.4670988718667)	(935.4166215465915, 88.45484396990594)
1	1	1.024944	-7.233608	(57.1908150949786, 51.383765538533396)	(908.8037183207852, 924.239157695396)
2	2	0.636115	468.713081	(101.54565380465604, 533.3077851463765)	(196.90855703046245, 593.969549852259)
3	3	-1.100406	800.770971	(212.43275057884958, 567.0087655385335)	(276.74726670788186, 496.236706715004)
4	4	4.114060	-1913.256173	(556.1827505788496, 374.91317730323925)	(647.1101699336883, 748.9940596561803)
5	5	6.037361	-3586.028190	(642.6746860627205, 294.0308243620628)	(724.7311376756238, 789.4352361267686)
6	6	-0.833215	867.857937	(17.27146025626891, 853.4670988718667)	(935.4166215465915, 88.45484396990594)

Rysunek 6 Odcinki w zestawie 3

	X	Y	ID Pierwszego odcinka	ID drugiego odcinka
0	713.985796	724.561730	5	1
1	470.945521	475.459099	1	0
2	648.255146	327.722073	5	0
3	617.012214	625.169255	4	1
4	562.150680	399.465599	4	0

Rysunek 7 Znalezione punkty przecięć w zestawie 3

Jak widać program znajduje wszystkie punkty przecięć pomiędzy odcinkami oraz nie dodaje do wyniku jednego punktu kilkukrotnie.

10. Podsumowanie i wnioski

Na podstawie wyżej przetestowanych zestawów można wnioskować, że program działa poprawnie. Laboratorium pokazuje, że dobór właściwych struktur danych jest często tak samo ważny jak sam algorytm. Dzięki strukturze sortedset zaimplementowany algorytm ma złożoność O((P+n)*logn), podczas gdy algorytm brutalny osiąga złożoność O(n^2). Warto zauważyć jednak, że przy niewydajnej strukturze stanów i zdarzeń, np. zwykłej liście sortowanej po każdym dodaniu elementu, złożoność algorytmu była by większa od rozwiązania brutalnego i wynosiła by O((P+n)*n*logn).